Phase 2 Report

Describe your overall approach to implementing the game.

We worked separately by dividing the work into three parts and created the class structures first. However, we do not have the experience for making a game. We tried to watch the tutorials, review the lecture notes, and learn some knowledge and algorithms from other games. We tried using branches to test our code before merging, and changing and modifying the codes to avoid merge conflicts. Pushes, commits and pulls also make the whole process simple and fast. To help us understand the codes with each other, we have written doc comments. After completing the implementations, we run the game to check for errors.

State and justify the adjustments and modifications to the initial design of the project (shown in class diagrams and use cases from Phase 1).

We remade the UML diagrams because we lost a group member during Phase 2 of the project (please see the new UML below).
Specifically, we changed:

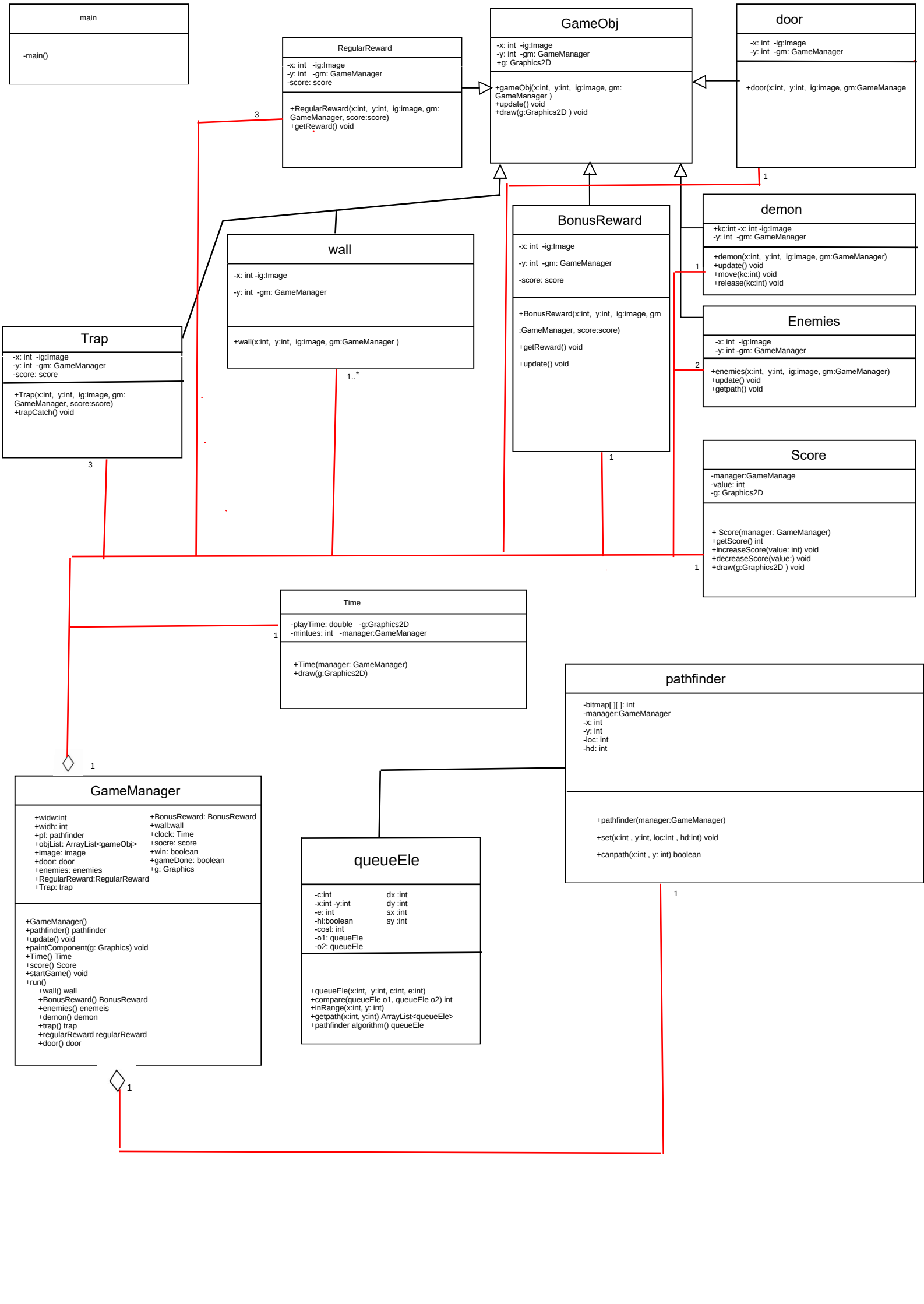
- the class name of main_character to Demon.
- the class name of enemy to Enemies.
- the class name of character to GameObj.
- the class name of game_timer to Time.
- the class name pf game_manger to GameManager.
- the class name of B_reward to BonusReward.
- the class name of Reward_ortrap to Trap.

We also added the classes RegularReward, Main, Wall, Pathfinder, and Door, and deleted some classes like Map, R_T_Type class, etc..

We created new relationships between classes using Generalization, Association and Aggregation.

We also added and modified some attributes and methods for the classes.
Then we add some UI classes based on UML during the project:

- CreditFrame
- EndFrame
- MenuFrame


main
-main()
RegularReward
-x: int -ig:Image
-y: int -gm: GameManager
-score: score
+RegularReward(x:int, y:int, ig:image, gm: GameManager, score:score)
+getReward() void
GameObj
-x: int -ig:Image
-y: int -gm: GameManager
+g: Graphics2D
+gameObj(x:int, y:int, ig:image, gm: GameManager)
+update() void
+draw(g:Graphics2D) void
door
-x: int -ig:Image
-y: int -gm: GameManager
+door(x:int, y:int, ig:image, gm:GameManage
wall
-x: int -ig:Image
-y: int -gm: GameManager
+wall(x:int, y:int, ig:image, gm:GameManager)
BonusReward
-x: int -ig:Image
-y: int -gm: GameManager
-score: score
+BonusReward(x:int, y:int, ig:image, gm
:GameManager, score:score)
+getReward() void
+update() void
demon
+kc:int -x: int -ig:Image
-y: int -gm: GameManager
+demon(x:int, y:int, ig:image, gm:GameManager)
+update() void
+move(kc:int) void
+release(kc:int) void
Enemies
-x: int -ig:Image
-y: int -gm: GameManager
+enemies(x:int, y:int, ig:image, gm:GameManager)
+update() void
+getpath() void
Trap
-x: int -ig:Image
-y: int -gm: GameManager
-score: score
+Trap(x:int, y:int, ig:image, gm: GameManager, score:score)
+trapCatch() void
Score
-manager:GameManage
-value: int
-g: Graphics2D
+ Score(manager: GameManager)
+getScore() int
+increaseScore(value: int) void
+decreaseScore(value:) void
+draw(g:Graphics2D) void
Time
-playTime: double -g:Graphics2D
-mintues: int -manager:GameManager
+Time(manager: GameManager)
+draw(g:Graphics2D)
pathfinder
-bitmap[ ][ ]: int
-manager:GameManager
-x: int
-y: int
-loc: int
-hd: int
+pathfinder(manager:GameManager)
+set(x:int , y:int, loc:int , hd:int) void
+canpath(x:int , y: int) boolean
GameManager
+widw:int
+widh: int
+pf: pathfinder
+objList: ArrayList<gameObj>
+image: image
+door: door
+enemies: enemies
+RegularReward:RegularReward
+Trap: trap
+BonusReward: BonusReward
+wall:wall
+clock: Time
+socre: score
+win: boolean
+gameDone: boolean
+g: Graphics
+GameManager()
+pathfinder() pathfinder
+update() void
+paintComponent(g: Graphics) void
+Time() Time
+score() Score
+startGame() void
+run()
+wall() wall
+BonusReward() BonusReward
+enemies() enemeis
+demon() demon
+trap() trap
+regularReward regularReward
+door() door
queueEle
-c:int
-x:int -y:int
-e: int
-hl:boolean
-cost: int
-o1: queueEle
-o2: queueEle
dx :int
dy :int
sx :int
sy :int
+queueEle(x:int, y:int, c:int, e:int)
+compare(queueEle o1, queueEle o2) int
+inRange(x:int, y: int)
+getpath(x:int, y:int) ArrayList<queueEle>
+pathfinder algorithm() queueEle
3
1
1..*
2

Explain the management process of this phase and the division of roles and responsibilities.

Our group began by redefining the scope and goals of the project to reflect the change in team size. We assigned roles and responsibilities based on individual skills and developed a project plan with a timeline for completing each task, resources needed, and potential risks that may arise during the project. The group members then began developing the maze game, coding the game mechanics, designing the user interface, and testing the game to ensure it was functional and met the desired specifications. Throughout the development process, our group regularly monitored their progress and made adjustments to their project plan as needed. Despite the change in team size, our group was able to effectively manage the project, ensuring its successful completion.

List external libraries you used, for instance for the GUI and briefly justify the reason(s) for choosing the libraries.

We have chosen Apache NetBeans to create GUI using JFrame over Visual Studio Code due to its ease of use and powerful features specifically designed for Java development. NetBeans offers a robust set of tools and libraries that simplify the creation of graphical user interfaces in Java applications, including support for Swing and JavaFX. Additionally, NetBeans provides features such as code completion, debugging, and code refactoring, making the development process more efficient and streamlined. No external libraries were required to create the GUI using JFrame and for any other parts of the project.

Describe the measures you took to enhance the quality of your code.

We took multiple measures to enhance the quality of our code. We wrote comments in addition to our Javadocs to clarify the purpose of the code that we added. We organized our code by grouping code used for similar purposes and using extra lines to separate those groups for more visual clarity; for example, in the GameManager class we put all the code that initialized our game objects together and separate from the screen settings. We also organized our code by utilizing more classes and files; instead of putting all our code into one file or one main class, we separated it as much as we could by, for example, having a separate class for our bonus and regular rewards.

In addition, we enhanced the quality of our code by making sure we all followed the same naming conventions: camelCase for our variables/methods, and PascalCase for our classes. We also made sure the names of our variables/methods/classes were descriptive (such as using “BonusReward” instead of “reward1” or “reward2”). We all looked over the code to make sure we understood it.

Discuss the biggest challenges you faced during this phase.

The biggest challenge that we faced during Phase 2 was how a group member, who had the most opinion on the specifics of the game, withdrew from the course without prior notice. We had to rework our plan for the game and redistribute the tasks, which meant that the project required more time from each of us that we had not initially expected or planned for. It was challenging as we all have other courses and work to attend to, not to mention other

commitments like family responsibilities and extracurriculars, that we had already scheduled around.

Another challenge we faced was figuring out how to work with the frontend/UI part of the project. It required a different approach from the other code which in turn required a bit of research and understanding of how to get started. We also found implementing the pathfinder challenging as it required lots of math and little details.